Certification Authorities Software Team (CAST)

Position Paper CAST-15

Merging High-Level and Low-Level Requirements

Completed February 2003

NOTE: **This position paper has been coordinated among the software specialists of certification authorities from the United States, Europe, and Canada. However, it does not constitute official policy or guidance from any of the authorities. This document is provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering for actual projects.**

Merging High-Level and Low-Level Requirements

Abstract:

Section 5.0 of DO-178B/ED-12B explains that when Source Code is generated directly from high-level requirements (HLR), then the high-level requirements are also considered low-level requirements (LLR), and the guidelines for low-level requirements also apply.

This part of section 5.0 addresses the case where the system-level requirements are "directly" highly refined (i.e., created in one refinement step); hence, allowing a single level of software requirements. However, applicants sometimes misuse this paragraph to justify combining high-level requirements (HLR) and low-level requirements (LLR) into the same data item. They establish development cycles that are intended to expand HLR with requirements built during the software design phase. Thus, they are "flattening" the requirements tree and merging all the requirements into a single data item, without establishing traceability between LLR and HLR.

In general, HLR represent "what" is to be designed and LLR represent "how" to carryout the design. Merging these two levels leads to several certification concerns, as described in this paper.

Key words:

High-level requirements, low-level requirements, requirements, design, traceability, verification

1 BACKGROUND

Section 5.0 (second paragraph) of DO-178B/ED-12B states that: "*Software development processes produce one or more levels of software requirements. High-level requirements are produced directly through analysis of system requirements and system architecture. Usually, these high-level requirements are further developed during the software design process, thus producing one or more successive, lower levels of requirements. However, if Source Code is generated directly from high-level requirements, then the high-level requirements are also considered low-level requirements, and the guidelines for low-level requirements also apply.*"

This part of section 5.0 addresses the case where the system-level requirements are "directly" highly refined (i.e., generated in one refinement step); hence, allowing a single level of software requirements. However, applicants sometimes misuse this paragraph to justify combining high-level requirements (HLR) and low-level requirements (LLR) into the same data item. They establish development cycles that are intended to expand HLR with requirements built during the software design phase. Thus, they are "flattening" the requirements tree and merging all the requirements into a single data item, without establishing traceability between LLR and HLR.

In general, HLR represent "what" is to be designed and LLR represent "how" to carryout the design. Merging these two levels leads to several certification concerns, as described in this paper.

2 DISCUSSION AND RECOMMENDATIONS

2.1 DO-178B/ED-12B Compliance Concerns

Applicants' motivation to expand HLR and merge with LLR during the design phase seems to be to keep all the requirements in a single data item. The resulting data item is the result of a development process that may not comply with DO-178B/ED-12B, since some objectives may not be satisfied. Combining HLR and LLR into the same data item is not a recommended practice because process assurance may be affected as follows (as a minimum):

- The complexity of the software requirement document is significantly increased. With no distinction between HLR and LLR, it is difficult to ensure compliance with traceability and verification objectives as defined in DO-178B/ED-12B. Traceability is "the evidence of an association between items, such as between process outputs, between an output and its originating process, or between a requirement and its implementation."
- Demonstration of the correct requirements "management" (particularly for large and complex systems and software) is much more difficult.
- The combination of the "what" (HLR) and "how" (LLR) may impact the relevance of the software life cycle data and the ability to verify the requirements.
- Derived requirements could get missed, which may have safety impacts (since derived requirements should be fed back to the system safety assessment process).
- Compliance demonstration with verification objectives during initial development is more difficult to achieve.
- Compliance demonstration with re-verification objectives for software modifications is more difficult, because change impact analysis will be more complex.
- The ability to manage and analyze the impact of changes/modifications to the airborne software becomes difficult or impossible.
- The consistency of the software requirements document is not ensured when modifying airborne software.
- The consistency and relevance of software requirements document with other development life cycle output data (source code, design architecture, system specification, etc.) is not ensured.

- Transition criteria is difficult or impossible to define and follow when HLR and LLR are combined.

Basically, compliance with DO-178B/ED-12B tables A-4, A-5 and A-6 objectives is weakened when HLR and LLR are merged into a single data item.

For compliance demonstration of DO-178B/ED-12B objectives, applicants should propose further justification and alternate means of compliance to satisfy the applicable objectives and to address the above concerns.

2.2 Verification Concerns

When airborne software components are large or complex, the software requirements process produces the HLR and the software design process produces the LLR and architecture. Thus, HLR and LLR are not the outputs of the same development processes (whatever the development process is) and cannot be considered as the same type of requirements (i.e., cannot be HLR and LLR at the same time). Therefore, there is no reason to reduce the amount of verification activities required for compliance demonstration with DO-178B/ED-12B section 6, depending on the development cycle followed. In addition, verification activities performed on HLR and LLR cannot be achieved at the same time, since the production processes are distinct.

Therefore, even in case where HLR and LLR are merged, the only way to perform verification activities seems to be the same way as it would have been performed in a standard development cycle (distinction maintained between HLR and LLR in the software life cycle data).

Furthermore, as documented traceability between HLR and LLR is required by DO178B/ED12B for verification purpose, loss of traceability is an issue. Alternative methods should be proposed to satisfy the same objectives as the one satisfied when maintaining documented traceability between HLR and LLR.

2.3 Re-verification Concerns (Modification of the Airborne Software)

When LLR and HLR are merged into one data item, without traceability between HLR and LLR, accurate and applicable re-verification is more difficult. A good change impact analysis is dependent on traceability in order to determine affected components during change. If the HLR to LLR trace is not documented, re-verification becomes difficult or impossible. Therefore:

- the change impact analysis method should establish relevance of the impact perimeter determination and the relevance of verification means (e.g., test subset) chosen to satisfy verification objectives for the modification. If the change impact analysis method is not satisfactory, alternative methods should be proposed to

address links between requirements embedded in the software requirements document.
- the level of assurance should be the same level as the one reached when HLR and LLR are traceable, maintained, and verified in separate data items.

Note: Notice 8110.85 further addresses change impact analysis guidelines.

3 CERTIFICATION AUTHORITIES' POSITION

As described in this paper, certification authorities do not recommend the merging of HLR and LLR into a single data item, because it makes satisfying the objectives of DO-178B/ED-12B difficult or impossible. HLR state what the system software is to do, and LLR states how the system software components are to do it. There are different verification approaches and objectives for HLR and LLR. For example, many of the HLR should be verified by the system level and hardware-software integration verification; whereas LLR typically cannot be verified at that level.

To use section 5 of DO-178B/ED-12B to justify merging of HLR and LLR such that visibility of parts of the development process is lost represents a misinterpretation of the original objective of section 5. Combining HLR and LLR complicates the ability to demonstrate compliance with DO-178B/ED-12B objectives, introducing confusion and increasing the certification task for software aspects.

Note: There may be some systems where the system-level requirements are "directly" highly refined (i.e., generated in one refinement step). In this case, a single level of software requirements may be feasible.

DO-178B/ED-12B established objectives to manage and reduce software development complexity, and it recommends separation of development processes and decomposition of associated output data. Merging HLR and LLR increases the complexity of software requirements document, software verification process, and software re-verification process. Merging HLR and LLR into a single data item for large and complex systems omits some of the objectives and benefits of applying DO-178B/ED-12B (i.e., assurance is lost). Therefore, the practice of combining HLR and LLR is not recommended, particularly for large, complex, or safety-critical airborne software. Combining HLR and LLR increases the complexity of software development assurance and the risk for undetected errors.

Although, compliance with DO-178B/ED-12B may eventually be shown, when merging HLR and LLR, the level of confidence that maintainability will be ensured over a long period of time is not equivalent to the one provided when following the DO-178B/ED-12B objectives. Even if compliance with the DO-178B/ED-12B objectives could be shown, maintainability may not be ensured for the entire life of the product. In actuality, if the HLR and LLR are merged into a single data item, the impact of such an alternate approach will need to be re-evaluated each time a change is made to the software (to ensure that the objectives of DO-178B/ED-12B section 6 are addressed).

Appendix A provides a copy of DO-248B/ED-94B frequently asked question (FAQ) #71. FAQ #71 provides additional insight into the objectives of DO-178B/ED-12B's requirement for traceability.

Appendix A - FAQ #71 (for reference purposes)

<u>FAQ #71</u>: What is the purpose of traceability, how much is required, and how is it documented? For example, is a matrix required or are other methods acceptable?

Reference: DO-178B/ED-12B: Sections 5.5, 6.2, 6.3, 6.4, 7.2.2, Annex A, and Annex B

Keywords: traceability

Answer:

Traceability is used to: (1) enable verification of implemented system requirements, high-level requirements, and low-level requirements; (2) to verify the absence of unintended function and/or undocumented source code; and (3) to provide visibility to the derived requirements. Traceability applies to both the verification and configuration management processes.

The format for documenting traceability is at the discretion of the applicant or developer. Whilst a matrix format is not required, it has proven successful for many past programs for capturing traceability data in a very usable and concise manner. In some respects, traceability may assist in tracking and controlling the entire software development and its status. Traceability may also assist in establishing the acceptability of derivations of previously developed and approved baselines.

The DO-178B/ED-12B glossary defines traceability as *"the evidence of an association between items, such as between process outputs, between an output and its originating process, or between a requirement and its implementation"* (reference Annex B). It facilitates evaluation of the processes and their outputs to facilitate the assurance of DO-178B/ED-12B objectives.

Traceability is bi-directional. DO-178B/ED-12B objectives call for both forward (top-down) and backward (bottom-up) traceability. The evidence of traceability may be in a format such that it can demonstrate both forwards and backwards traceability.

Forward traceability shows where each of the requirements, including derived requirements, trace down through the design to their implementation and provides a means to demonstrate that all requirements have been implemented. Forward traceability may also support change analysis by showing those lower-level components (e.g., design, code, test cases) that are affected when a requirement is changed.

Backward traceability shows the origin of the implementation traced up through the design to the requirements. Additionally, the traceability evidence may help demonstrate that nothing has been included in the implementation that is not traceable to the requirements.

To understand traceability, one should consider all aspects of traceability referred to in the following discussion, which lists some of the benefits of traceability and refers to sections of DO-178B/ED-12B that discuss its various objectives, guidance, and activities. Traceability is used in conjunction with the integral processes to provide the following benefits:

- To ensure completeness between development data elements, for example, between system requirements and software requirements, between software requirements (high-level) and software design (low-level), and between software design elements and source code components (reference Section 5.5 of DO-178B/ED-12B).
- To ensure completeness for verification coverage objectives, for example, between software requirements and the verification procedures and verification results for establishing requirements-based test coverage (reference DO-178B/ED-12B Sections 6.2a, 6.2b, 6.3.1f, 6.3.2f, 6.3.4e, 6.4.4.1a, 6.4.4.2.b, and Annex A objectives).
- To facilitate the certification liaison process and certification authority visibility and confidence by providing a usable means for assessing the status and completeness of software life cycle data and its processes throughout the project (for on-site certification authority and designee reviews) and for follow-on certification efforts (reference Sections 9.2 and 12.1.1d of DO-178B/ED-12B).
- To enable the developer to perform change impact analysis; to readily identify software life cycle data that may be affected by changes, both during the development and for post-certification changes; and to determine the scope and required effort to “re-verify” the affected software data items.
- To assist new project personnel implementing software changes to understand the relationships between the software life cycle data during the potentially long service life of many airborne systems and equipment.

With respect to the level of traceability, the developer should select an appropriate level of granularity for which traceability will be provided for each type of element. For example, requirements – one or several functionally related; source code – module or procedure or line of code; verification tests – procedure or case. Since traceability supports efficient access to related elements of different data items, the level of granularity should be selected that best supports effective and efficient access. If the granularity is too coarse, it may be difficult to identify and access all relevant elements efficiently. On the other hand, if the granularity is too fine, maintaining the traceability data itself may become unwieldy and burdensome. Some combination of approaches could be used, for example, using a matrix for general location of related elements and using embedded features such as code comments for correlation at a finer level.